DETECT MEMORY LEAK BY ANALYZING HEAP

J1 WORKFLOW

MEMORY LEAK

- A memory area which is no longer needed is not release.
- Causes out of memory or crash when there’s not enough memory.
- Hard to detect.
- Hard to reproduce.

COMMON SOLUTIONS

Solutions	Pros	Cons
Static check tools (Cppcheck, compiler warnings…)	Simple, Fast, Easy to use	Can miss run-time leaks False positives Cannot detect special cases
Memory debuggers (Valgrind, mtrace…)	Run-time check Precise, adequate results	Bad performance Big memory overhead
Logger (DLT, console log, …)	Less overhead Detect many kinds of issues	Cannot put log everywhere Missing logs

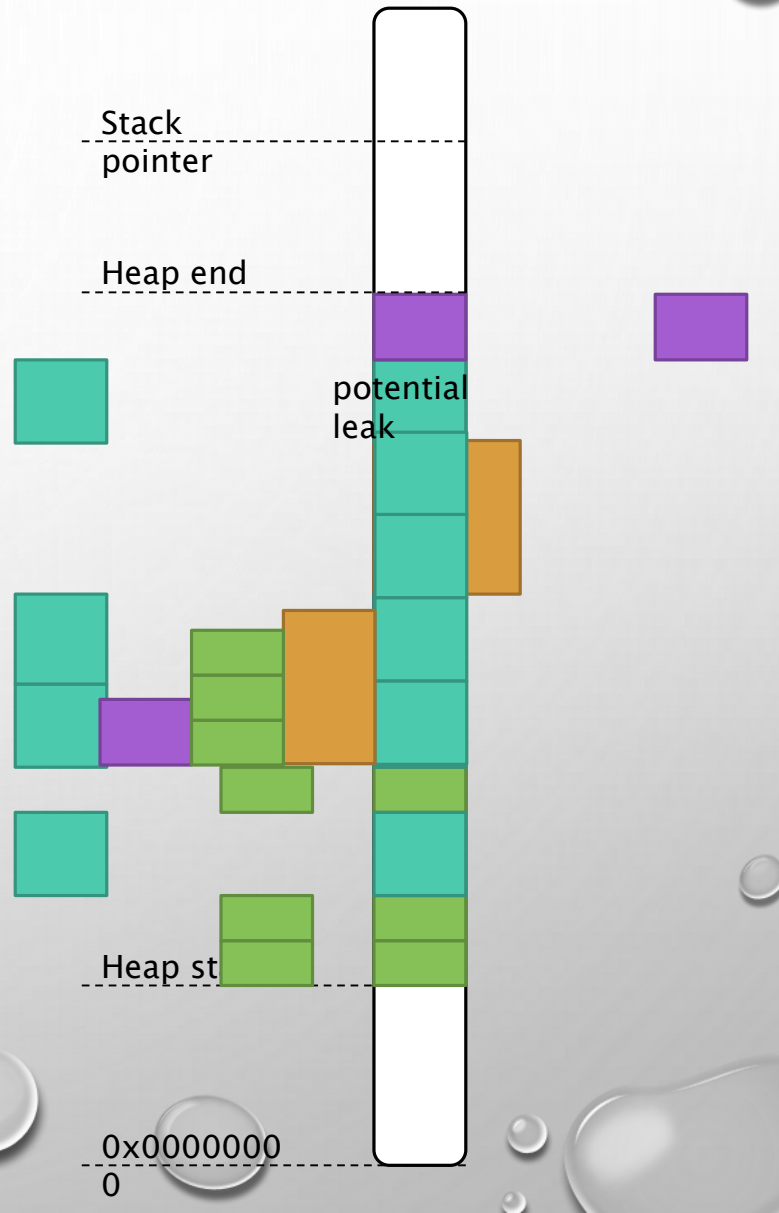
HEAP ANALYSIS

MOTIVATION

- Dynamically allocations are allocated in heap
- Memory leaks are duplicated blocks
- The most duplicated blocks in heap are most likely to be leaks.

CONCEPTUAL WORK FLOW

- Find heaps of the process
- Collect all memory chunks of the heaps
- Classify the chunks based on their size
- Determine the potential leaks based on their count and their total size
- Investigate the chunk structure to find where in the source code it is created *(hardest part)*



J1 WORK FLOW

- Use the QNX memory analysis tool to generate a report about the heap.
- Analyze the report to determine the leakage, using several techniques:
  - Generate a histogram of chunks.
  - Track chunks count throughout several time points.
  - …
- Determine the types of the leaked chunks, and then find the leakage in the source code.

DETERMINE THE TYPE

- Build a data type table.
- Investigate a suspicious chunk.
  - Find a string
  - Check the vtable.